

Volker Roth

Rechnersicherheit, SoSe 21

Übung 01

TutorIn: Oliver Wiese

Tutorium 02

Materialien: Latex, VSC, Skript

6. Mai 2021

1 (Un-)salted password hashes

We used the simple brute force method and tried all possible password variations. With the given alphabet which contain 26 characters, this took your program $62^4 = 14.776.336$ many iterations.

```

5 #alphabet
6 string_alphabet = string.ascii_lowercase + string.ascii_uppercase + '0123456789'
7 alphabet = list(string_alphabet)

9 #create dictionary of file
10 dict = {}
11 seed = ''
12 with open("sha2pwd.txt", "r") as file:
13     #strip unnecessary leading and ending chars
14     seed = file.readline().rstrip("Seed: ").rstrip("\n")
15     for line in file:
16         hash = line.rstrip("\n")
17         dict[hash] = 0

19 #password test each possible combination of alphabet with 4 chars
20 for char1 in alphabet:
21     for char2 in alphabet:
22         for char3 in alphabet:
23             for char4 in alphabet:
24                 password = char1 + char2 + char3 + char4
25                 salt = seed+password
26                 hash = hashlib.sha256(salt.encode()).hexdigest()
27                 try:
28                     test = dict[hash]
29                     dict[hash] = password
30                     #print(hash, password)
31                 except:
32                     pass

34 #result to file
35 with open('result_single_process.txt', 'w') as file:
36     file.write(json.dumps(dict))

```

Moreover the also implemented the idea above with multithreading and multiprocessing. As expected the implementation with multithreading changed nothing in terms of execution time. The multiprocessing implementation however gave us quite a boost. Below is a snippet of the multiprocessing code.

```

1 def run(tid, task, seed, hash_dict):
2     #password test each possible combination of alphabet with 4 chars
3     for char1 in task:
4         for char2 in alphabet:
5             for char3 in alphabet:
6                 for char4 in alphabet:
7                     password = char1 + char2 + char3 + char4
8                     salt = seed+password
9                     hash = hashlib.sha256(salt.encode()).hexdigest()
10                    try:
11                        test = hash_dict[hash]
12                        #save password to file
13                        with lock:
14                            with open('result_multi_processing.txt', 'a') as file:
15                                file.write(hash + ': ' + password + '\n')
16                    except:
17                        pass

19 def main():
20     #split into equal tasks of size TASK_SIZE for processes
21     tasks = []
22     single_task = []
23     for char in string_alphabet:
24         single_task.append(char)
25         if len(single_task) == TASK_SIZE:
26             tasks.append(single_task)
27             single_task = []
28     if single_task != []:
29         tasks[-1] = tasks[-1] + single_task

31     #delete file if it exists else create it
32     with open('result_multi_processing.txt', 'w') as file:
33         pass

35     #start processes
36     process_list = []
37     for i in range(len(tasks)):
38         process = Process(target=run, args=(i, tasks[i], seed, hash_dict))
39         process.start()
40         process_list.append(process)

42     #wait for all processes to finish
43     for process in process_list:
44         process.join()

```

```

~/Dokumente/rechnersicherheit/rechnersicherheit-sose-21/u2$ python3 password_multi_processing.py
enter number of processes: 1
--- 13.36160659790039 seconds ---
~/Dokumente/rechnersicherheit/rechnersicherheit-sose-21/u2$ python3 password_multi_processing.py
enter number of processes: 2
--- 6.8104634284973145 seconds ---
~/Dokumente/rechnersicherheit/rechnersicherheit-sose-21/u2$ python3 password_multi_processing.py
enter number of processes: 4
--- 3.667797088623047 seconds ---
~/Dokumente/rechnersicherheit/rechnersicherheit-sose-21/u2$ python3 password_multi_processing.py
enter number of processes: 16
--- 1.8293890953063965 seconds ---
~/Dokumente/rechnersicherheit/rechnersicherheit-sose-21/u2$

```

2 Password breaches

a)

The screenshot shows a crossword puzzle interface. At the top, there's a 'Password popularity' filter with tabs from 1-100 to 901-1000. The 101-200 tab is selected. The puzzle grid is on the left, with some letters filled in: 'm', 'q', 'w', 'h', 'e', 'e', 's', 'e', 'k', 'r', 'e', 't', 'y', 'y', 'a', '1', '2', '3', '4', '5', '6', '7', '8', '9', '1', '0', 'h', 'o', 't', 'm', 'a', 'i', 'l'. To the right, there are two columns of clues: 'Across' and 'Down'. Each clue is in a button-like box with a dropdown arrow. The 'Across' clues are: 6: pYTSIq/hxxk=, 7: FNr/AqTT8bvioxG6CatHBw==, 8: 0TU3uKPpPhg=, 9: jBp+HwtWWT+YRcfahC+Tdw==, 10: Kg4:xXFwXDM=. The 'Down' clues are: 1: f8SITDaggtc=, 2: g2B6PhWEH366cdBSCql/UQ==, 3: d8IJq+5iC0nINF+G++BX7W==, 4: g+/hUkh3HrbSPm/keox4IA==, 5: rNhveK0RH5Q=, 7: bx7K/eMGcgw=.

b)

Passwords are not salted, so duplicates stand out.

Even worse, the use of the block mode encrypts blocks of bytes separately, so you can see duplicate parts of the passwords. You can see this in the comic, for example the later half of the passwords with hint „Best TOS Episode“ and „sugarland“ are the same, and the first half of the passwords with hints „Best TOS Episode“ and „sexy earlobes“ are the same. That’s why the comics calls it a crossword puzzle.

Password hints should maybe also be encrypted?

3 Project

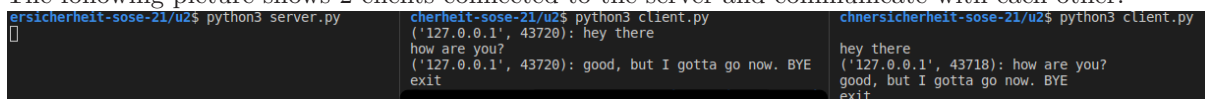
In the following you will see a snippet of python code from the server side.
We used threads and sockets for the task

```
30     def run(self):
31         #incoming msg thread
32         if self.tid == 0:
33             #while client_socket is alive
34             while not self.client_socket._closed:
35                 #recv msg from client fo size DATASIZE
36                 msg = self.client_socket.recv(DATASIZE)
37                 #if client disconnected
38                 if msg == bytes(0):
39                     #remove client from connected_clients and exit
40                     with lock:
41                         connected_clients.remove(self.client_socket)
42                     #write disconnect to log
43                     with open('log.txt', 'a') as file:
44                         date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
45                         file.write(date + ': ' + str(self.client_address) + '
46 disconnected\n')
47                     #stop thread
48                     sys.exit()
49                     #store msg in msg_buffer
50                     with lock:
51                         msg_buffer.append(myMsg(msg, self.client_socket, self.client_address))
52
53         #outgoing msg thread
54         if self.tid == 1:
55             while True:
56                 time.sleep(1)
57                 #if msg to send not empty and clients are available
58                 if msg_buffer != [] and connected_clients != []:
59                     #send msg to each client
60                     for s in connected_clients:
61                         #if client is still connected and client is not the sender of the
62                         #msg
63                         if not s._closed and s != msg_buffer[0].socket:
64                             #send msg to client
65                             s.sendall(msg_buffer[0].get_final_msg_bytes())
66                             #remove msg from bugger
67                             with lock:
68                                 mymsg = msg_buffer.pop(0)
69                                 #write msg to log
70                                 with open('log.txt', 'a') as file:
71                                     date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
72                                     file.write(date + ': ' + 'msg send: "' + mymsg.
73 get_final_msg_bytes().decode('utf-8') + '" \n')
```

Here is the snippet from the client side.

```
16 def run(self):
17     #incoming msg thread
18     if self.tid == 0:
19         #while server_socket is alive
20         while not self.server_socket._closed:
21             #waits for msg from server of size DATASIZE
22             msg = self.server_socket.recv(DATASIZE)
23             #print msg
24             print(msg.decode('utf-8'))
25
26     #outgoing msg thread
27     if self.tid == 1:
28         #while server_socket is alive
29         while not self.server_socket._closed:
30             #wait for keyboard input
31             msg = input()
32             if msg == 'exit':
33                 os._exit(1)
34             #send msg
35             server_socket.send(bytes(msg, 'utf-8'))
```

The following picture shows 2 clients connected to the server and communicate with each other.



The screenshot shows three terminal windows. The left window is the server, titled 'sicherheit-21/u2\$ python3 server.py'. It shows two connections from '127.0.0.1' at ports 43724 and 43726. The middle window is a client titled 'sicherheit-21/u2\$ python3 client.py'. It shows the client sending 'hey there' and 'how are you?' to the server at port 43720, and receiving 'good, but I gotta go now. BYE' from the server. The right window is another client titled 'sicherheit-21/u2\$ python3 client.py'. It shows the client sending 'hey there' and 'how are you?' to the server at port 43718, and receiving 'good, but I gotta go now. BYE' from the server.

This is the log file produced by the server.

```
1 05/05/2021 10:26:38: new connection from('127.0.0.1', 43724)
2 05/05/2021 10:26:48: new connection from('127.0.0.1', 43726)
3 05/05/2021 10:26:54: msg send: "('127.0.0.1', 43724): hey there"
4 05/05/2021 10:27:02: msg send: "('127.0.0.1', 43726): how are you?"
5 05/05/2021 10:27:17: msg send: "('127.0.0.1', 43724): good, but I gotta go now. BYE"
6 05/05/2021 10:27:18: ('127.0.0.1', 43724) disconnected
7 05/05/2021 10:27:21: ('127.0.0.1', 43726) disconnected
```